Planning and Implement Branching Strategies

Marcel de Vries
CTO

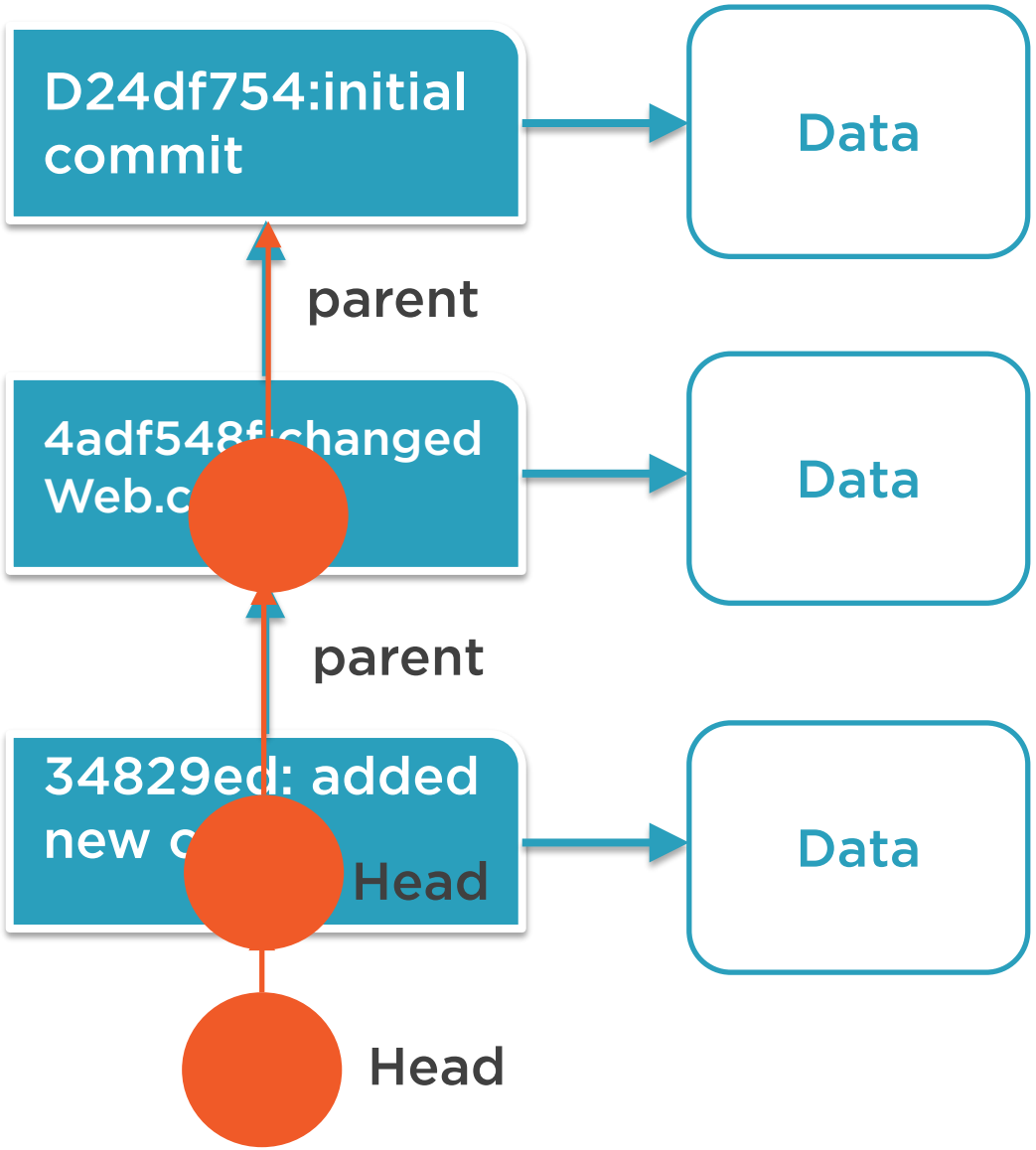
@marcelv https://Fluentbytes.com

Outline

- Understanding Git
- Using Pull requests
- Branching and Merging
- Using Git Tags
- Branching strategies
- Implementing and enforcing automation
- Summary

Understanding Git

Git Fundamentals



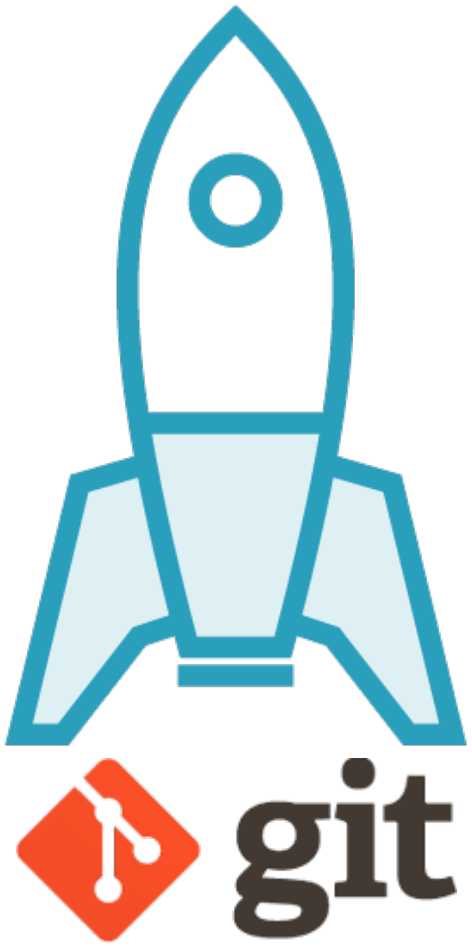
A Git commit is a node in a graph

Every commit has a pointer to its parent

References make commits reachable

- Head, Tag, Branch

Basic Commands



Clone

- Start working locally

Stage, Commit

- Change files and commit

Push

- Share with others

Pull

- Get work from others

Using Pull Requests

Pull Requests

Introduced by GitHub

Ensure code is reviewed before committed

Ensure quality gates

- CI/CD Build
- Unit Test succeed
- Etc.

Approve code before committed to main branch

Branching and Merging

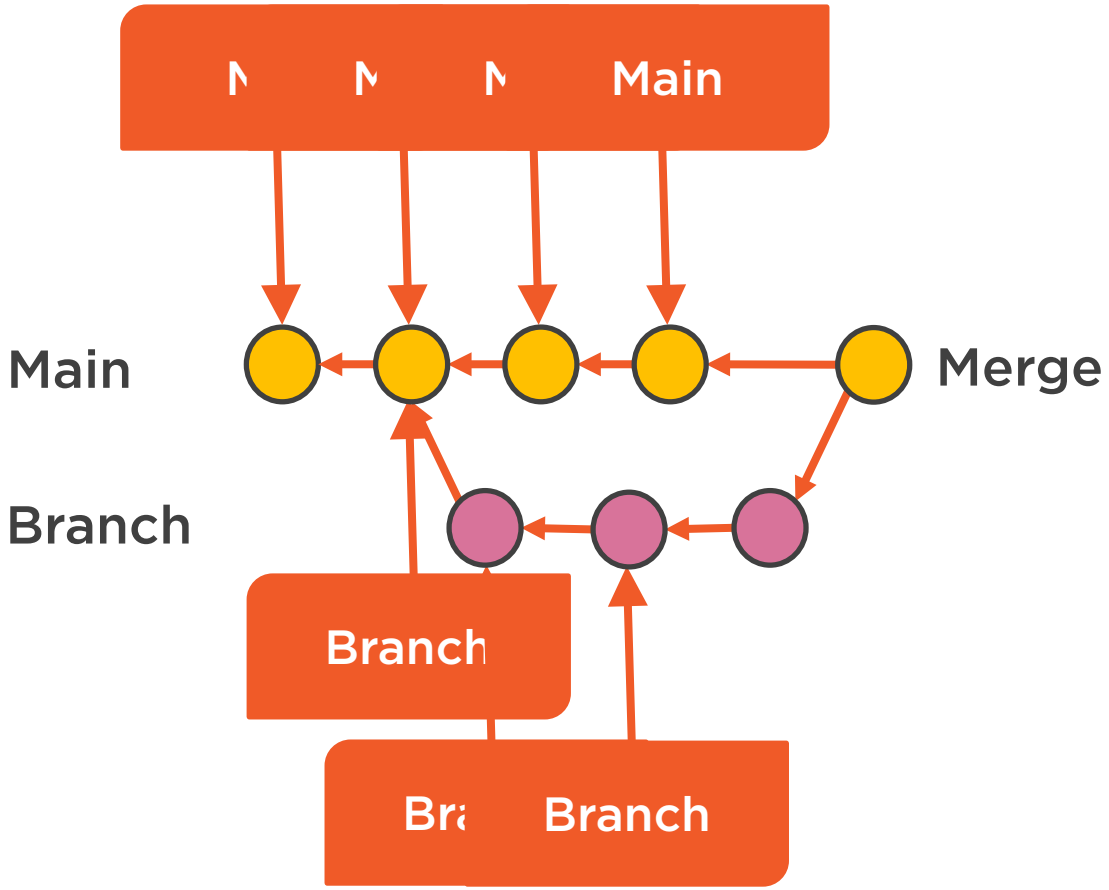
What Is a Branch?

A branch enables you to work on code isolated from others

When you branch you intend to merge

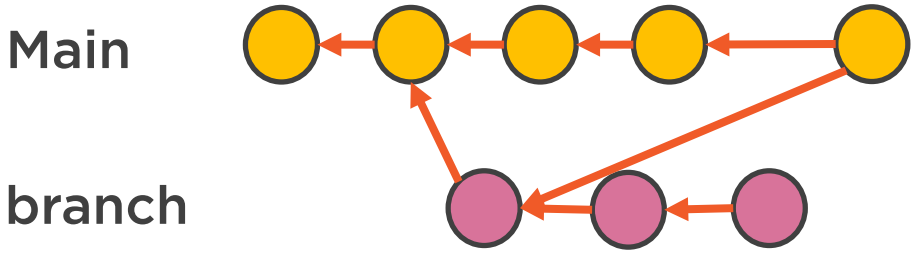
Branches have a traceable history to their parents

Branch and Merge

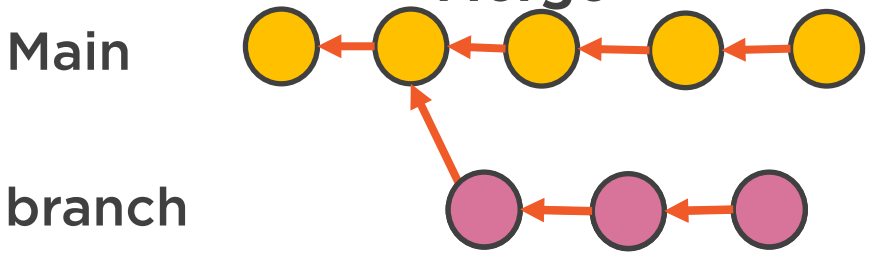


Merge and Rebase

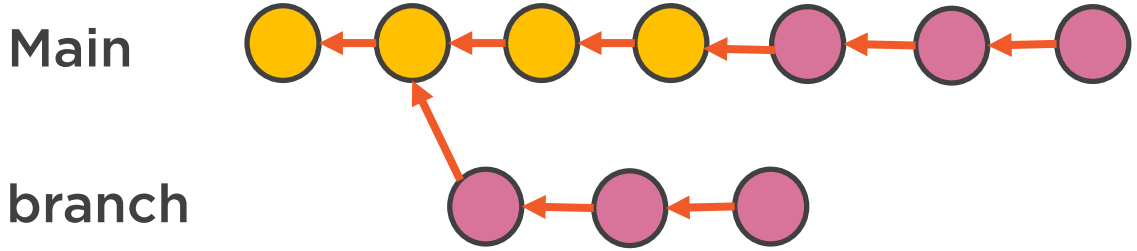
Squash & Merge



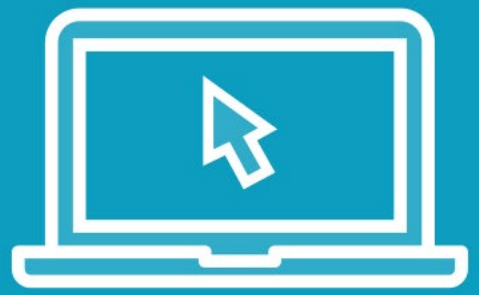
Fast forward Merge



Rebase

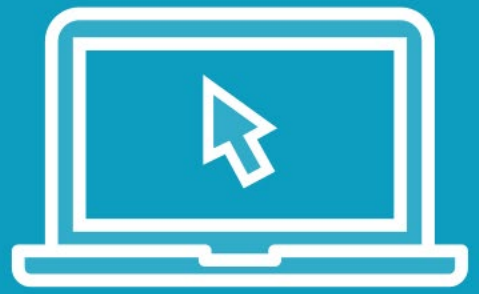


Demo



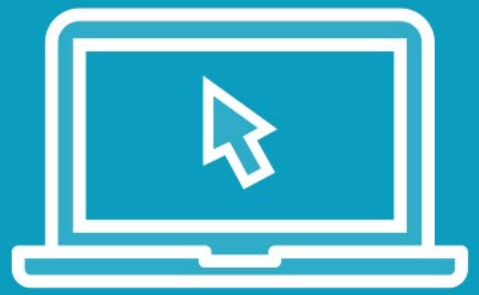
Branching and Merging: Merge

Demo



Branching and Merging: Fast Forward

Demo



Branching and Merging: Rebase

Using Git Tags

Using Git Tags

Git has the ability to tag specific points in a repository's history as being important

- Often used to mark release

Lightweight

- Pointer to specific commit

Annotated

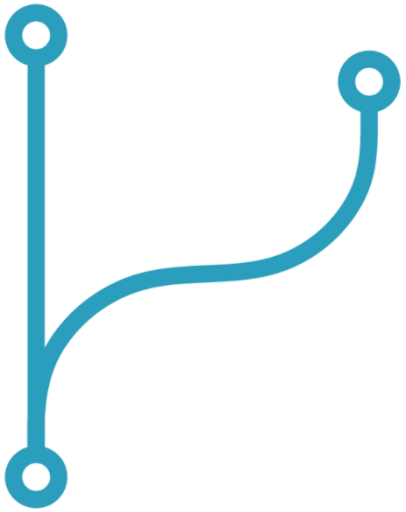
- Contain more information such as the tagger, message, and date

Branching Strategies

Most Common Git Branching Strategies

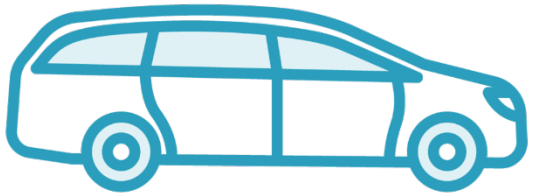
Git flow

Low deployment frequency



Git Hub Flow

High deployment frequency



Trunk Based Development

High Deployment frequency

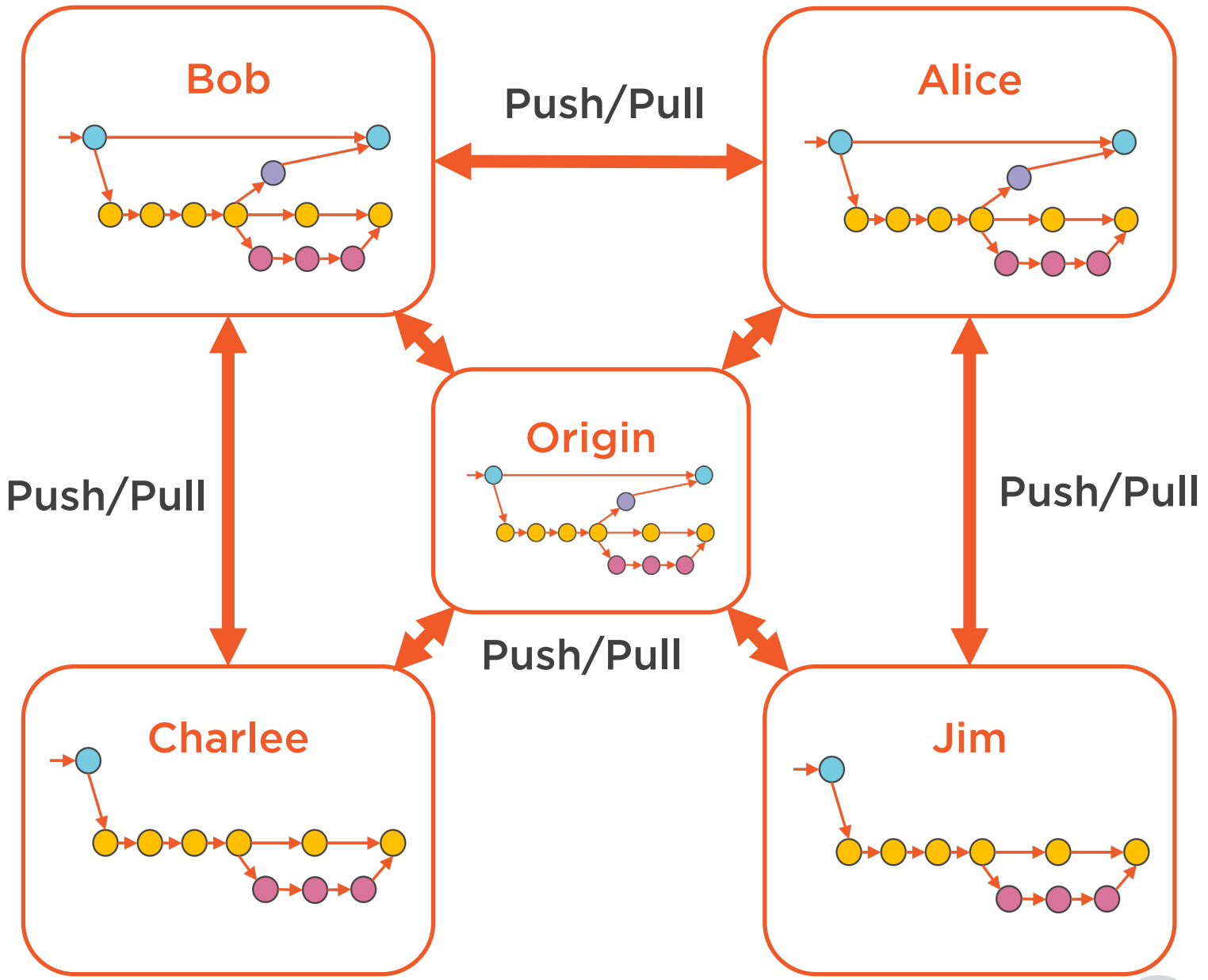
The Git Flow Branching strategy

Introduced in 2010 by Vincent Driessen

Use a central server model, called Origin

Every team member works on a clone

Team members can push/pull from team members



http://nvie.com/posts/a-successful-git-branching-model/

The Main Branches

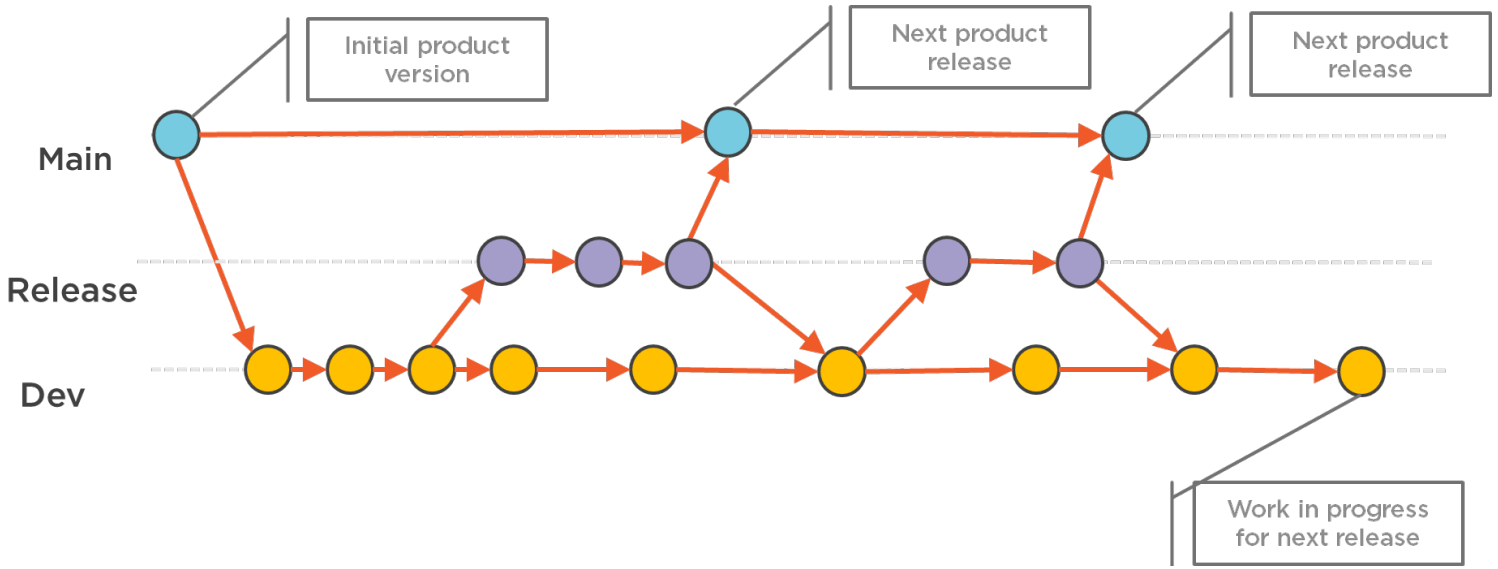
You always release from **Main**

Work is done on **Dev**

Release branch created to stabilize for release

Dev is where we work, but we strive for a stable always releasable product!

Main is always stable and ready to release



Supporting Branches

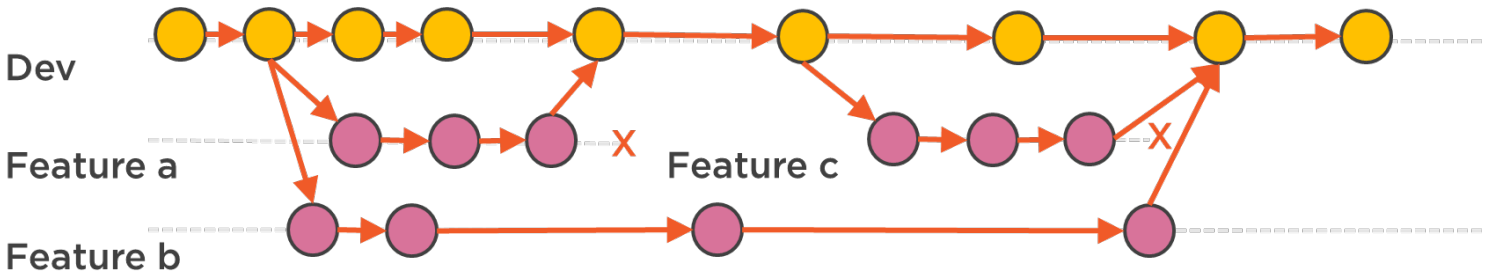
When we develop a feature you develop this on a Feature branch

Team members can work on that branch together

Peer to peer or via published branch on server

Feature branch always *merge* to Dev

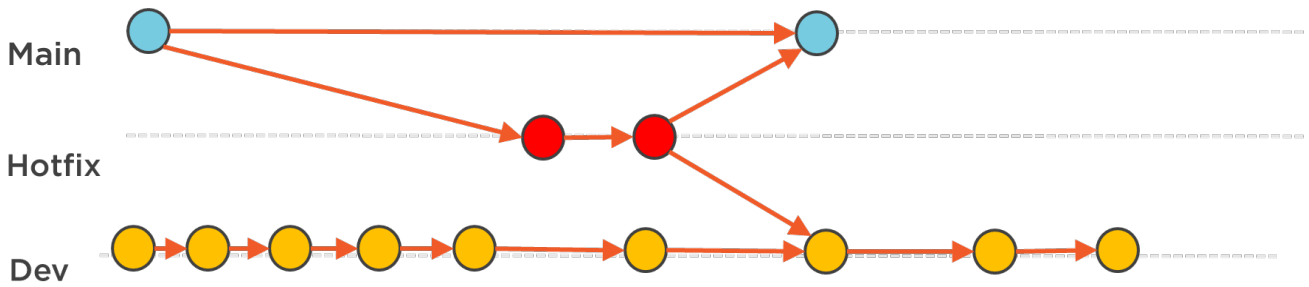
Feature branch is *deleted* after merge



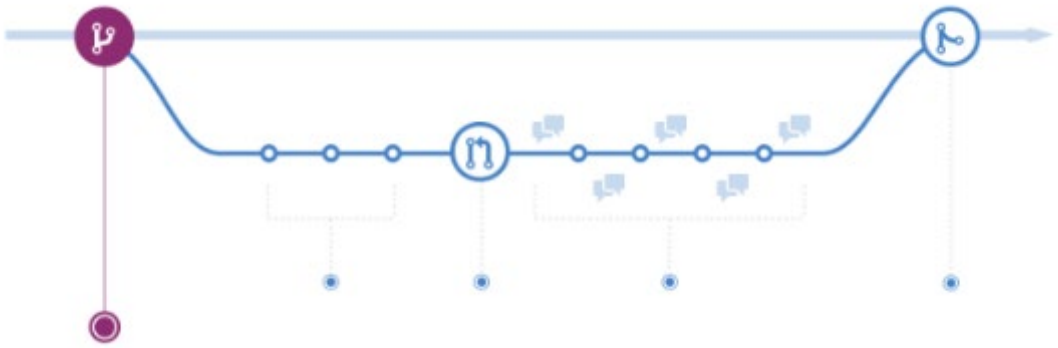
Hotfix Branches

You always create a **Hotfix** branch for critical production issues

You merge the **Hotfix** to **Main** and **Dev** when done



Git Hub Flow

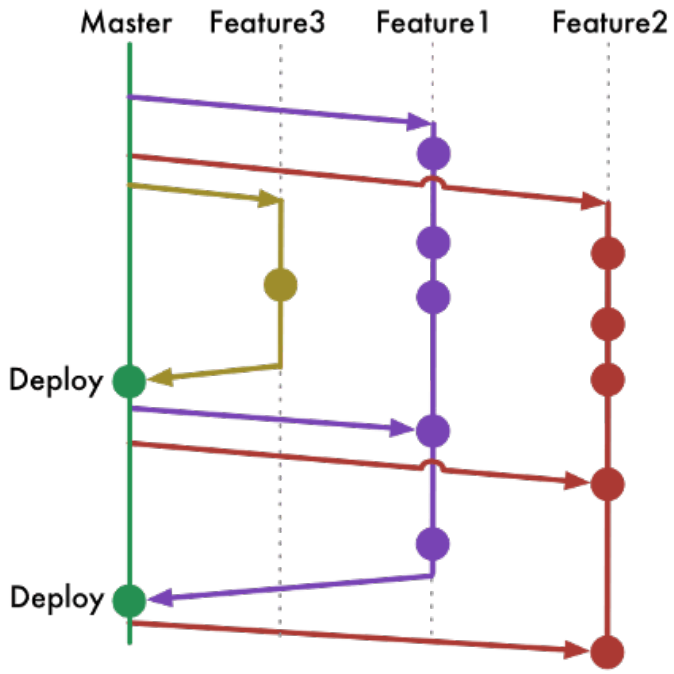


Best suited for continuous deployment

Release multiple times a day

Delivery of Software as a Service (24x7)

Trunk Based Development

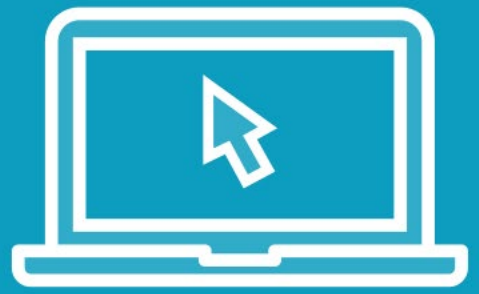


Almost the same as GitHub flow

- Release always from main
- Merge to main before you deploy

Use feature toggles to carry cross multiple integrations to main

Demo



Using Git Flow Branching Strategy

Implementing and Enforcing Automation

Pull Request Approval Flows

Conversation on suggested code change

- Code review & pre-validation

Azure DevOps

- Use pipelines to automate validation

GitHub

- Use actions to automate validation

Branch Policy

Enforce what is needed before you accept a merge

- Use of pull requests

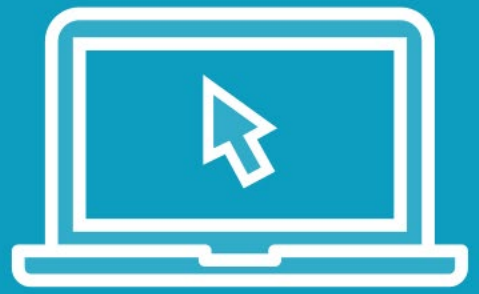
Automatic CI build for QA

Minimum # of reviewers

Enforce 4 eyes principle

Security checks / scanning

Demo



Setup a Branch Policy

Summary

- Understanding Git
- Using Pull requests
- Branching and Merging
- Using Git Tags
- Branching strategies
- Implementing and enforcing automation